Annales Univ. Sci. Budapest., Sect. Comp. 17 (1998) 293-306

REPRESENTATIION AND QUERY LANGUAGES OF FUZZY RELATIONAL DATABASES

T. Nikovits and A. Kiss (Budapest, Hungary)
D. Chretien (Versailles, France)

Abstract. In real life data cannot always be defined precisely. The existing database management systems cannot handle these imprecise pieces of information. There have been several attempts to extend the capabilities of databases in this respect.

In the paper we propose an extended relational data model based on fuzzy sets. In this model the values of a relation can be not only atomic, but they can be fuzzy sets as well. These fuzzy sets can represent our imprecise knowledge. We define an extended tuple relational calculus for the model. Then we show how the model can be implemented on an existing relational database management system, how the elements of the model can be represented by a system of ordinary tables, and how the standard SQL language can be extended to support the new model.

1. Introduction

In the real world there exist data which cannot be precisely defined. Ordinary database systems cannot handle these ambiguous data. In the last few years there have been numerous efforts to extend the underlying data models of these systems to improve their capabilities of handling imprecise information. A database system based on such a new model could handle vague information and could answer queries like this:

"Find all people who are middle aged and whose salary is not very high".

Supported by the Hungarian National Science Grant (OTKA), Grant Nr. 2149, and the Copernicus LPDB Project, Project Nr. 93:6638.

There are several theoretical models on which this problem can be based. The most promising from these seems to be the theory of fuzzy sets (Zadeh [10]). A fuzzy set is the generalization of the characteristic function of a set, that is a function from the basic set to [0,1]. In a point x the function value $f(x)$ expresses how much this point can be considered to be an element of the set.

In the last decade there have been several attempts to generalize the relational data model using Fuzzy Set Theory (Buckles-Petry [2], Prade-Testemale [5], Zemankova [11], Umano [8]). The common feature in these models is that they all extend the classical relational model of Codd, so the basic elements of the model remain the tables. In the extensions the attributes can have special data types e.g. set type (Buckles [2]), fuzzy set type (Vila [9]). Another possibility is that we associate a number between 0 and 1 with each tuple. The so defined fuzzy relations can be handled mathematically well (Kiss [3], Raju [6]), but they have less practical importance. Some authors use a mixed model taking ideas from both mentioned extensions (Umano [8]).

Different query languages, too, can be found in the literature for the different models. There are extensions of relational algebra and relational calculus as well. A general principle in these query languages is the *extension principle* (Novak [4]) which says how to apply an ordinary function or operation to fuzzy sets.

As we can see there is not a unique fuzzy data model. We will propose one which is rich enough to express the new concepts and simple to implement. Our model will allow fuzzy sets as values in the relations. In addition we introduce some more functions and relations which will help us to handle the vagueness in the data.

The rest of the paper is organized as the following. In Section 2 we give the elements of the model, in Section 3 we define a formal query language, fuzzy tuple relational calculus for the model. In Section 4 we show how to represent the model in a relational database management system and finally in Section 5 we introduce an extended SQL language that we will call FSQL.

2. Fuzzy relational data model

In this section we introduce the concepts of the fuzzy data model.

Definition 2.1. Let U be a set. The fuzzy set A over U is a function from U to [0,1]. $A : U -> [0,1]$ ($A(x)$ - often denoted by Ax - is the membership grade of x). We will denote the set of fuzzy sets over U with $P(U)$.

Definition 2.2. Let $D_1, D_2, ..., D_n$ be arbitrary domains. A fuzzy relation R over these domains is a subset of the Cartesian product $P(D_1) \times P(D_2) \times ... \times P(D_n)$. A fuzzy database is a collection of fuzzy relations.

A tuple of a relation has the form $(a_1, a_2, ..., a_n)$ where each a_i is an element of $P(D_i)$.

Relations usually have attribute names associated with the domains and we denote the scheme of a relation with $R(A_1, A_2, ..., A_n)$ where A_i-s are the attributes. Attribute names must be different within a relation, while among the domains repetition is allowed.

We note that the classical relations are special cases of fuzzy relations while an element of D_i can be considered a special fuzzy set over D_i. So $D_i \subset P(D_i)$.

Example 2.1. Let R be a relation in which we store the names, ages, salaries and other data of some people. The attributes of the relation are Name, Age, Salary etc. A tuple of the relation can be the following:

Name	Age	Salary	...
John Taylor	Middle-age	About 30000	...

Here the age and salary columns have fuzzy sets as their values, they can be given by the following functions:

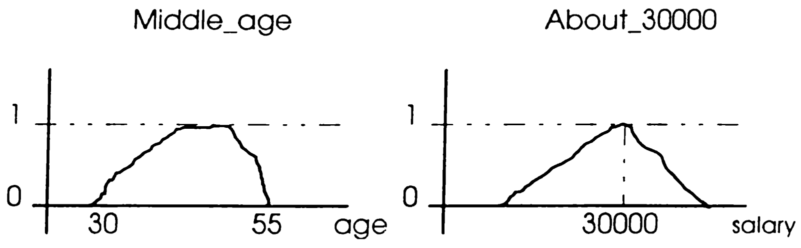

Figure 2.1. Values in a relation expressed by fuzzy sets

With the aid of these fuzzy sets we can express our imprecise knowledge about the age and salary of the person.

We introduce some other functions which will be part of the data model as well. One group of these functions is modifiers and the other is similarity relations.

Definition 2.3. A modifier is a function from [0,1] to [0,1]. A k-ary similarity relation is a function from $D_{i1} \times D_{i2} \times \ldots \times D_{ik}$ to [0,1] where $D_{i1}, D_{i2}, \ldots D_{ik}$ are among the domains of the relations.

Example 2.2. An example for a modifier can be the function named *very* which is defined by $very(x) = x^2$. With the aid of similarity relation $similar\_age(x, y)$ we can decide how two people can be considered having the same age. We will use these functions in the queries. The two abovementioned functions could be used in the following query: Get the names of people whose salary is very high and whose age is about the same as that of Mr. Taylor.

The modifiers enrich the model in that we can express personal opinions with them (e.g. rather, very etc.) The similarity relations can be considered the extensions of the equality relation.

3. Query languages of the model

For the relational data model there are two well-known formal query languages: relational algebra and relational calculus. Relational calculus has two different forms according to how the user sees the data: tuple relational calculus and domain relational calculus. In the following we will give an extension of the tuple relational calculus to our model.

3.1. Fuzzy tuple relational calculus

Fuzzy tuple relational calculus is an extension of tuple relational calculus. The expressions of the calculus have the form $\{t | F(t)\}$, where F is a first order formula whose only free variable is t. t is a tuple variable for which different tuples can be substituted. We will denote the i-th component of a tuple variable with $t[i]$ and the dimension of t with $t^{(n)}$. The main point in the extension is that here the truth value of a formula can be not only true (1) or false (0) but any value between 0 and 1. This value can be compared with a previously given number λ to decide which tuples will belong to the result relation of the query. We therefore define two different truth values for a formula, the previously mentioned will be the fuzzy truth value of the formula (FT) which can be any number between 0 and 1, and we define the truth value of the formula (T) which can be true (1) or false (0). After substituting a tuple into the formula this tuple will belong to the result if the truth value of the formula becomes 1. The formulas can be the following:

Atomic formulas:

- If R is an n-ary relational symbol, then $R(t^{(n)})$ is a formula and the occurrence of t is free in it. The fuzzy truth value $FT(R(t^{(n)}))$ and the truth value $T(R(t^{(n)}))$ of the formula are 1 iff $t \in R$, otherwise both are 0.

- $t[i] == u[j]$ is a formula where the occurrence of t and u is free. The fuzzy truth value $FT(t[i] == u[j])$ and the truth value $T(t[i] == u[j])$ of the formula are 1 iff $t[i]$ and $u[j]$ are identical fuzzy sets, that is they are identical as functions. Otherwise both are 0.

- $t[i] == k$ is a formula where k is a fuzzy set. The occurrence of t is free in the formula. The fuzzy truth value $FT(t[i] == k)$ and the truth value $T(t[i] == k)$ of the formula are 1 iff $t[i]$ and k are identical fuzzy sets, that is they are identical as functions. Otherwise both are 0.

- $(t[i]\ \theta\ u[j],\ \lambda)$ is a formula where θ is a relational symbol from the set $\{<, >, <=, >=, =, !=\}$ and $\lambda \in [0,1]$. The occurrences of t and u are free in the formula. We define the truth values of the formula based on the extension principle. Let $\theta(x,y) = 1$ if $x\theta y$ holds, 0 otherwise. We substitute the fuzzy sets $t[i]$ and $u[j]$ into the function $\theta(x,y)$. The result will be a fuzzy set over the set $\{0,1\} : \theta(t[i], u[j]) = \{c/1, d/0\}$. We fix a function f from $[0,1] \times [0,1]$ to [0,1], let f be the following: $f(c,d) = c$. Now we define the fuzzy truth value of the formula: $FT((t[i]\ \theta\ u[j],\ \lambda)) =$ $= f(\theta(t[i], u[j]))$. The truth value of the formula $T((t[i]\ \theta\ u[j], \lambda)) = 1$ if $FT((t[i]\ \theta\ u[j], l)) >= \lambda$, 0 otherwise.

- $(t[i]\ \theta\ k,\ \lambda)$ is a formula where θ is a relational symbol from the set $\{<, >, <=, >=, =, !=\}$, k is a fuzzy set and $\lambda \in [0,1]$. The occurrence of t is free in the formula. We define the truth values of the formula similarly as in the previous case. The fuzzy truth value of the formula $FT((t[i]\ \theta\ k,\ \lambda)) = f(\theta(t[i], k))$. The truth value of the formula $T((t[i]\ \theta\ k],\ \lambda)) = 1$ if $FT((t[i]\ \theta\ k,\ \lambda)) >= \lambda$, 0 otherwise.

- Let s be an n-ary similarity relation, then $(s(A_1, ..., A_n), \lambda)$ is a formula where each A_i is a fuzzy set or a component of a tuple variable and $\lambda \in$ $\in [0,1]$. The occurrences of the tuple variables in the formula are free. We define the truth values of the formula in the following way. $s(A_1, ..., A_n)$ is a fuzzy set over [0,1], so we fix a function g from $P([0,1])$ to [0,1], let g be the following: $g(A) = A(1)$. Then the fuzzy truth value of the formula $FT((s(A_1, ..., A_n), \lambda)) = g(s(A_1, ..., A_n))$. The truth value of the formula $T((s(A_1, ..., A_n), \lambda)) = 1$ if $FT((s(A_1, ..., A_n), \lambda)) >= \lambda$, otherwise it is 0.

We can build other formulas with the aid of logical connectors, modifiers and quantifiers.

- If F and G are formulas then $(F \wedge G, \lambda)$ is a formula, too, where $\lambda \in [0,1]$. The occurrences of the tuple variables in the formula do not change. The fuzzy truth value of the formula $FT((F \wedge G, \lambda)) = \min(FT(F), FT(G))$. The truth value of the formula $T((F \wedge G, \lambda)) = 1$ if $T(F) = 1$, $T(G) = 1$ and $FT((F \vee G, \lambda)) >= \lambda$, otherwise it is 0.

- If F and G are formulas then $(F \vee G, \lambda)$ is a formula, too, where $\lambda \in [0,1]$. The occurrences of the tuple variables in the formula do not change. The fuzzy truth value of the formula $FT((F \vee G, \lambda)) = \max(FT(F), FT(G))$. The truth value of the formula $T((F \vee G, \lambda)) = 1$ if $T(F) = 1$ or $T(G) = 1$ and $FT((F \vee G, \lambda)) >= \lambda$, otherwise it is 0.

- If F is a formula and m is a modifier then $(m(F), \lambda)$ is a formula, too, where $\lambda \in [0,1]$. The occurrences of the tuple variables in the formula do not change. The fuzzy truth value of the formula $FT((m(F), \lambda)) = m(FT(F))$. The truth value of the formula $T((m(F), \lambda)) = 1$ if $FT((m(F), \lambda)) >= \lambda$, otherwise it is 0.

- If F is a formula then $(\neg F, \lambda)$ is a formula, too, where $\lambda \in [0,1]$. The occurrences of the tuple variables in the formula do not change. The fuzzy truth value of the formula $FT((\neg F, \lambda)) = 1 - FT(F)$. The truth value of the formula $T((\neg F, \lambda)) = 1$ if $T(F) = 0$ and $FT((\neg F, \lambda)) >= \lambda$, otherwise it is 0.

- If t has a free occurrence in the formula F then $(\forall t\ F, \lambda)$ is a formula, too, where $\lambda \in [0,1]$. All the occurrences of the variable t in this formula are bound. The fuzzy truth value of the formula $FT((\forall t\ F, \lambda)) =$ $= \inf\{FT(F) \mid \text{for all the possible tuples } t\}$. The truth value of the formula $T((\forall t\ F, \lambda)) = 1$ if $T(F) = 1$ and $FT((\forall t\ F, \lambda)) >= \lambda$ for all the possible tuples t, otherwise it is 0.

- If t has a free occurrence in the formula F then $(\exists t\ F, \lambda)$ is a formula, too, where $\lambda \in [0,1]$. All the occurrences of the variable t in this formula are bound. The fuzzy truth value of the formula $FT((\exists t\ F, \lambda)) =$ $= \sup\{FT(F) \mid \text{for all the possible tuples } t\}$. The truth value of the formula $T((\exists t\ F, \lambda)) = 1$ if there exists a tuple t for which $T(F) = 1$ and $FT((\exists t F, \lambda)) >= \lambda$, if there is no such tuple then the truth value is 0.

Formulas can be constructed only in the abovementioned way. If a tuple variable has a free occurrence in a formula then we call this variable free in the formula, otherwise it is called bound. To simplify the notation we can write F instead of the formula $(F, 0)$.

Example 3.1. We give some example queries and express them in fuzzy tuple relational calculus. The scheme of the relation R is the same as in the previous example.

Name	Age	Salary	...

[Query 1] Who are the people whose age is 30 with certainty level 0.7 ?

$$\{s^{(1)} \mid \exists t(R(t) \wedge (t[2] = 30, 0.7) \wedge (t[1] == s[1]))\}$$

[Query 2] Who are the people whose age is young with certainty level 0.8 or whose salary is very high with certainty level 0.9 ?

$$\{s^{(1)} | \exists t(R(t) \wedge (t[1] == s[1]) \wedge ((t[2] = young, 0.8) \vee (very(t[3] = high), 0.9)))\}$$

[Query 3] Get the pairs of people whose age is similar with certainty level 0.8 ?

$$\{u^{(2)} \mid \exists t\ \exists s(R(t) \wedge R(s) \wedge (\text{similar\_age}(t[2], s[2]), 0.8) \wedge (u[1] ==$$

$$== t[1]) \wedge (u[2] == s[1]) \wedge \neg(u[1] == u[2]))\}$$

4. Representation of the model

In the previous sections we gave the formal definitions of the fuzzy relational data model and defined a formal query language for this model. We would like to implement this model over ordinary relational database management systems, so we would like to represent all the elements of the model with ordinary tables. We will not exploit the specialities of any RDBMS, we will base on the common core of them. That is why we will restrict the previously defined model and will allow only special fuzzy sets, special modifiers and special similarity relations to occur.

Fuzzy relational data model is an extension of the relational data model, where the columns of a relation can have a new data type, the *fuzzy type*. We can store imprecise information in the columns having this type. The values in the fuzzy type columns can be some fuzzy sets. As we saw, the ordinary integer, float, date and character type data (as elements of the underlying domains) can be considered as special cases of fuzzy sets, so in a fuzzy type column we can store ordinary values as well as fuzzy sets. Now we will allow only special fuzzy sets as values which can be characterized by finitely many parameters.

So in a relation we can have the following data types:

1. The standard data types (integer, float, date, character ... etc.)
2. The fuzzy version of the data types (fuzzy integer, fuzzy float, fuzzy date, fuzzy char, ...etc.)

We can store the following data in the columns of a relation:
In the columns, whose data type is some standard data type, we can store the ordinary values allowed in the underlying relational system.
In fuzzy columns we can have the following special fuzzy sets depending on the underlying domain of the column:

1. Trapezoid fuzzy sets, if the underlying domain is ordered.

2. Fuzzy sets given by continuos linear sections, if the underlying domain is ordered.

3. Discrete possibility distributions on any given underlying domain.

We consider the underlying domain as ordered if it is integer, float or date.

Example 4.1. We will have a relation EMPLOYEE whose columns will be Name, Salary, Age and Language (meaning the languages spoken by the person). The data types of the columns are the following: Name column is character type, Salary and Age columns are fuzzy integers and Language column is fuzzy character.

The following table shows one possible instance of the relation.

EMPLOYEE

Name	Salary	Age	Language
George Scott	28000	42	{1/English, 0.4/French, 0.8/Japanese}
John Taylor	{ 1/25000, 0.9/26000, 0.8/27000 }	29	German
Paul Smith	trapezoid(21000, 24000, 28000, 34000)	34	{1/Russian, 1/Spanish}
Adam Clark	linear(0/20000, 0.4/25000, 1/27000, 0.7/32000, 0/35000)	56	English

In the Salary column we can see an example for each possible value a fuzzy integer column can have. The salary of Mr. Scott is given by an ordinary integer value that can be stored in any RDBMS. In this case we have precise information about the data. The salary of Mr. Taylor is given by a discrete possibility distribution. Its meaning is that we have only vague information about the salary. There are more than one possible value and each value has a corresponding possibility. In this case we have only finitely many possible values for the salary. In the last two tuples we have infinitely many possible values for the salary, every value having a possibility. The values and the corresponding possibilities are given by the two special fuzzy sets. We can see these fuzzy sets in the pictures below:

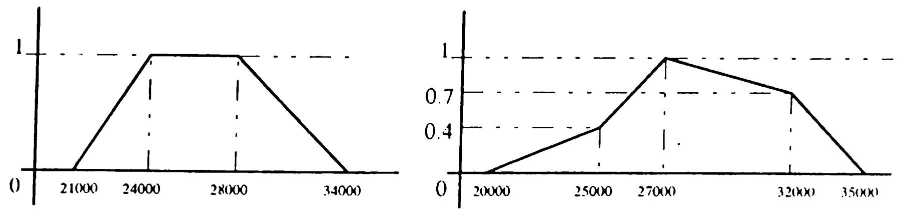

Figure 4.1. Trapezoid and linear type fuzzy sets

All columns in the model having ordered underlying domain can have values given by one of these four possibilities. The Language column shows us what we can store in a fuzzy character column. In the case of Mr. Taylor and Mr. Clark we have precise information about their language knowledge. In the two other tuples the discrete possibility distributions give us the finitely many possible values and their corresponding possibilities.

Similarity relations:

We will allow only similarity relations with two arguments in this restricted model. With these relations we can give how much two different values in the database can be considered approximately equal or similar to each other. We will use these similarity relations in queries.

When we define the similarity relation we have to give the domain type on which the relation is based. Both arguments of the relation should come from this type. The measurement of the similarity is a number between 0 and 1. So the similarity relation is a function having two arguments of the same type and yielding a number between 0 and 1. This function can be defined in the following ways: If the function has only finitely many discrete pairs of elements as argument, then we should give all the pairs and the corresponding similarity values for them. In this case the relation have to be symmetric, that is it should give the same similarity value for the pairs (A, B) and (B, A). The relation have to be reflexive in the sense that for every element A it should give 1 as a similarity value for the pair (A, A).

When the underlying domain is ordered we can give the similarity degree of two values as a function of the difference between the two values. Only some special functions will be allowed, these are the step functions. We define the function with (a, b) pairs, where a denotes the maximal difference between the two values and b gives the similarity degree for these values. We can compare the age of people with the similarity relation *similar_age*. If the difference of ages between two people is under 3 years then we consider them as of the same age (possibility value 1). As the difference of ages grows, we consider them less and less the same age, so the possibility value decreases. We can express this with the following step function:

$$similar\_age(x, y) = \begin{cases} 1, & \text{if } |x-y| \leq 3 \\ 0.7, & \text{if } 3 < |x-y| \leq 6 \\ 0.4, & \text{if } 6 < |x-y| \leq 8 \\ 0, & \text{otherwise.} \end{cases}$$

We can define the step function with the following parameters:

$similar\_age(step, integer, 1/3, 0.7/6, 0.4/8)$.

Modifiers:

Modifiers are functions from [0,1] to [0,1]. They can modify the possibility value belonging to a value. Because of the finite representation we will allow only functions defined by finitely many linear sections. An example for this can be the modifier *very* which is defined by the following parameters and can be seen in the picture below:

$very = \{0/0,\ 0.2/0.4,\ 0.4/0.6,\ 1/1\}$

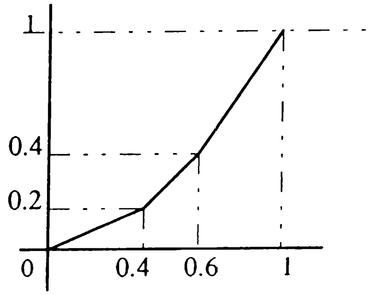

Figure 4.2. Modifier given by linear sections

So far we gave all the additional objects that are part of our extended fuzzy data model. This data model will be implemented over an ordinary RDBMS, so the definitions of the fuzzy sets, modifiers and similarity relations and all the information about the relationship among the different fuzzy objects will be represented by ordinary tables. Here we give the system of tables by which we can represent this information. The set of these additional tables will be called the meta-database and the tables themselves are the meta-tables. The meta-database in our model consists of the following meta-tables:

FUZZY COLUMNS

Table name	Column name	Column id	Column type

FUZZY OBJECTS

Column id	Object name	Object id	Object type

MODIFIERS

Object id	Value	Modified value

SIMILARITY DISCRETE

Object id	Object 1	Object 2	Value

SIMILARITY STEP

Object id	Difference	Value

TRAPEZOID

Object id	Value 1	Value 2	Value 3	Value 4

LINEAR

Object id	Value	Possibility

DISCRETE

Object id	Value	Possibility

The name of every column which allows some fuzzy handling will be stored in the FUZZY_COLUMNS meta-table. The different fuzzy objects will be stored in different meta-tables. The relationship between the objects will be created through the identifiers.

5. An extended SQL language: FSQL

We restricted our fuzzy model in the previous section to be able to implement it over relational databases. We need a query language to create, delete and update these new elements, too. In the field of relational databases SQL is the language which became standard in the latest years. All the market leader systems support some version of SQL. That is why we chose this language as the basis of the query language of our model. We defined some extensions which enables the language to handle the fuzzy objects as well. We will refer to this language as FSQL. The commands of the language can be divided into two subsections - as is the case with SQL - Data Definition Language (DDL) and Data Manipulation Language (DML). In the following we will show the main concepts of the extensions.

5.1. Data definition language

The data definition language contains the statements by which we can create and drop the elements of the model. We preserved the DDL commands of SQL and defined new commands for the fuzzy objects. All our extensions are based on the previously presented representation. We give here only one example for the command by which we can create a new similarity relation.

CREATE SIMILARITY name <data_def>

<data_def> ::- (DISCRETE, <discrete_def>) | (STEP, <step_def>)

<discrete_def> ::- <possibility>/<value> <value>
{, <possibility>/<value> <value> }

<step_def> ::- <possibility>/<value> {, <possibility>/<value> }

5.2. Data manipulation language

With the commands of the data manipulation language we can insert, delete, update and query the elements of the model. The syntaxes of the standard SQL commands became a little more complex to be able to handle the fuzzy elements as well. The syntax and semantics of the SELECT statement changed most significantly. The FSQL SELECT is based on the fuzzy tuple relational calculus. When we evaluate the search condition given after the WHERE clause, then the result will not be true (1) or false (0) as in the ordinary case, but it can be an arbitrary number between 0 and 1. To decide which tuples will qualify the user can give a λ level and the tuples having truth value greater than this λ will belong to the result. Instead of the ordinary comparison predicate we can use the following construction.

<comparison> ::- <expression> θ <expression> [WITH λ] |

similarity_relation(<expression> , <expression>) [WITH λ] |
modifier(<expression> θ <expression>) [WITH λ] |
modifier(similarity_relation(<expression> , <expression>)) [WITH λ]

Example 5.1. We give an example Select statement here for the query that we saw in Example 3.1. Who are the people whose age is young with certainty level 0.8 or whose salary is very high with certainty level 0.9 ?

```
SELECT Name
FROM Employee
WHERE Age = young WITH 0.8
OR very(Salary = high) WITH 0.9
```

6. Conclusion

In the paper we gave a possible extension of the relational data model, based on fuzzy sets, by which we can handle imprecise information. We defined a formal query language for the model, the fuzzy tuple relational calculus. Then we turned to the practical aspects of implementation. We showed how this model can be represented by a system of relations. Finally we extended the standard query language of relational systems to this model. Our restricted model can be the basis of further work when we will investigate additional possibilities of how fuzzy sets can be used in modelling imprecise concepts. This model was defined so that it could be implemented on relational databases. In the future, however, new data models, especially object oriented ones, will become more and more widespread in the field of databases, too. Our future work will be focussed on the possible extensions of these models as well.

References

[1] **Bosc P. and Pivert O.**, About Equivalences in SQLf, A Relational Language Supporting Imprecise Querying, *IFES'91 Brussels,* vol. 3, 1991, 309-319.

[2] **Buckles B.P. and Petry F.E.**, A Domain Calculus for Fuzzy Relational Databases, *Fuzzy Sets and Systems,* **29** (1991), 327-340.

[3] **Kiss A.**, λ-Decomposition of Fuzzy Relational Databases, *Annales Univ. Sci. Bud. Sect. Comp.*, **12** (1991), 133-142.

[4] **Novak V.**, *Fuzzy Sets and their Applications*, IOP Publishing Ltd., 1986.

[5] **Prade H. and Testemale C.**, Generalizing Database Relational Algebra for Treatment of Incomplete or Uncertain Information and Vague Queries, *Information Science*, **34** (1984), 115-143.

[6] **Raju K. and Majumdar A.**, The Study of Joins in Fuzzy Relational Databases, *Fuzzy Sets and Systems*, **21** (1987), 19-34.

[7] **Ullman J.D.**, *Principles of Database and Knowledge-Base Systems, Vol.I*, Computer Science Press, 1988.

[8] **Umano M. and Ezawa Y.**, Implementation of SQL-type Data Manipulation Language for Fuzzy Relational Databases, *Proc. Conf. on Fuzzy Systems*, Japan, 1991, 276-280.

[9] **Vila A., Medina J.M., Pons O., Cubero J.C., Prados M. and Diaz J.**, A Knowledge Representation Model for Fuzzy Databases, *ACM Trans. on Database Systems* (submitted)

[10] **Zadeh L.A.**, Fuzzy Sets as a Basis for a Theory of Possibility, *Fuzzy Sets and Systems*, **1** (1978), 3-28.

[11] **Zemankova-Leech M. and Kandel A.**, *Fuzzy Relational Databases a Key for Expert Systems*, TUV, Rheiland, 1984.

T. Nikovits and A. Kiss
Department of General Computer Science
Eötvös Loránd University
VIII. Múzeum krt. 6-8.
H-1088 Budapest, Hungary
nikovits@ullmann.inf.elte.hu
kiss@ullmann.inf.elte.hu

D. Chretien
Laboratoire PRiSM
Université de Versailles
Saint-Quentin-en-Yvelines
45, Av. des Etats-Unis
78035 Versailles Cedex
France
Didier.Chretien@prism.uvsq.fr